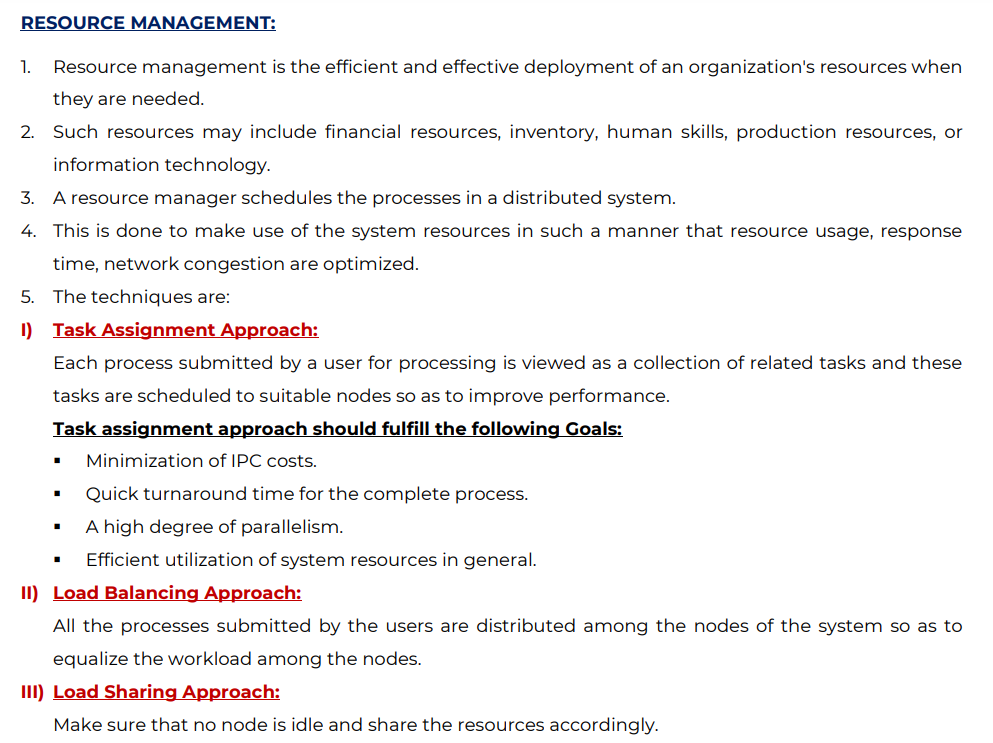
**Mod 4 : Resource Management**



A distributed system, also known as distributed computing, is a system with multiple components located on different machines that communicate and coordinate actions in order to appear as a single coherent system to the end-user.

It contains various nodes that are separated but are connected to each other through a network.

Resource Sharing: Computers in distributed systems shares resources like hardware (disks and printers), software (files, windows and data objects) and data. Hardware resources are shared for reductions in cost and convenience. Data is shared for consistency and exchange of information.

Process: A process is defined as an entity which represents the basic unit of work to be implemented in the system. To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

A heuristic, or heuristic technique, is any approach to problem-solving that uses a practical method or various shortcuts in order to produce solutions that may not be optimal but are sufficient given a limited timeframe or deadline.

Scheduling: The method used to determine which of several processes, each of which can safely have a resource allocated to it, will actually be granted use of the resource.

Scheduling overhead means: The time taken by dispatcher to move the process from ready state to running state.

1. State the desirable features of global scheduling algorithm?

a. No A Priori Knowledge about the processes:

The working of the scheduling algorithm is based on the information and the resource requirements of the processes. Due to which, it poses an extra burden on the users who must provide this information before submitting their processes to execution.

b. Dynamic in Nature:

The process assignment by the scheduling algorithm should be dynamic in nature i.e. based on the current load of the system and not on some static policy. The algorithm should also have the flexibility to migrate the process more than once, because the initial decision to place a process on a particular node may change after some time to adapt to the new system load.

c. Decision Making Capability:

Using the heuristic method, a problem in the system is solved by using less computational efforts and the solution is a near-optimal result.

d. Balanced System Performance and Scheduling Overhead

An algorithm that gives near-optimal performance/result is required. Having more information counts for intelligent decisions but scheduling overhead increases. It happens that the scheduling frequency is low because of the cost of gathering and processing extra information. The overhead increases as the amount of global state information increases i.e. CPU load.

e. Stability

Processor thrashing i.e. the fruitless migration of processes should be avoided. For example, we have nodes n1 and n2 which are loaded with processes and n3 is an idle node. When we offload a portion of their work to node n3 without being aware that the offloading was also performed by other nodes then node becomes overloaded. Due to this, it may start transferring its processes to other nodes. The main reason for this is that the decision of scheduling a process is happening independently at each node.

f. Scalability

A scheduling algorithm should scale well as the number of nodes/networks in the system increases. A scheduling algorithm that schedules by first inquiring the workload from all the nodes and then selecting the most lightly loaded node has poor scalability. This will work fine only when there are few nodes in the system. This is because the inquirer receives a flood of replies simultaneously from the nodes and the time required to process the reply messages for making a node selection is too long as the number of nodes (N) increase.

g. Fault Tolerance

A good scheduling algorithm should be able to handle the system if one or more than one nodes crash. In order to have better fault tolerance, algorithms should decentralize the decision-making process and must consider only the nodes that are available.

h. Fairness of Service

The global scheduling policy which blindly attempts to balance the load from the nodes are not good from the point of view of fairness of service. This is because, in any load-balancing scheme, heavily loaded nodes will obtain all the benefits while lightly loaded nodes will suffer poorer response time. Hence, load-balancing has to be replaced by the concept of load sharing, that is a node will share some of its resources as long as its users are not significantly affected.

2. Code Migration & Issues

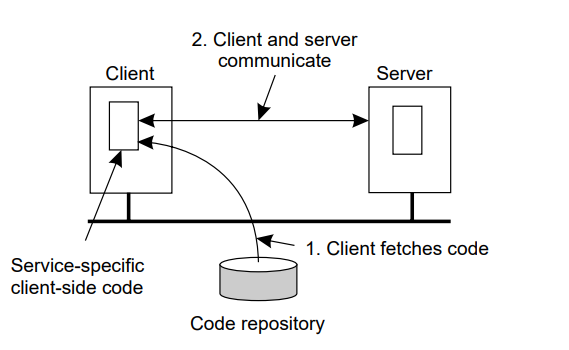
Code migration is when programs are moved from one machine to another, often moving parts of its execution environment along with it. The intent is for the transferred program to continue execution on the target machine.

Traditionally, code migration in distributed systems took place in the form of process migration in which an entire process was moved from one machine to another. Moving a running process to a different machine is a costly and intricate task, and there had better be a good reason for doing so. That reason has always been performance. The basic idea is that overall system performance can be improved if processes are moved from heavily-loaded to lightly-loaded machines. Load is often expressed in terms of the CPU queue length or CPU utilization, but other performance indicators are used as well.

Consider, for example, a client-server system in which the server manages a huge database. If a client application needs to do many database operations involving large quantities of data, it may be better to ship part of the client application to the server and send only the results across the network. Otherwise, the network may be swamped with the transfer of data from the server to the client. In this case, code migration is based on the assumption that it generally makes sense to process data close to where those data reside.

This same reason can be used for migrating parts of the server to the client. For example, in many interactive database applications, clients need to fill in forms that are subsequently translated into a series of database operations. Processing the form at the client side, and sending only the completed form to the server, can sometimes avoid that a relatively large number of small messages need to cross the network. The result is that the client perceives better performance, while at the same time the server spends less time on form processing and communication.

Besides improving performance, there are other reasons for supporting code migration as well. The most important one is that of flexibility. If code can move between different machines, it becomes possible to dynamically configure distributed systems.



Here the server is responsible to provide the clients implementation, when the client binds to the server. The advantage is that the client does not need to install any software. The software can be moved in as when necessary and discarded when it is not needed.

Code Migration Issues :

1. Lack of Planning

It may not seem like a technical error, however most code migration issues can be due to lack of proper planning, an adequate data migration plan. With trending technologies changes in the system are added quite fast. People who are responsible to plan before migrating the code should not neglect these parameters, otherwise the system can crash and could lead to project failures etc. Therefore planning and having the knowledge about the target system is quite crucial.

2. Compatibility Issues

When a code from system A is migrated to system B, we also need to make sure that is code in system A when it is migrated to system B is compatible to the framework of system B. Knowing the prerequisites and how the code should be adjusted in system B should also be looked upto. If not, the code can sometimes crash the whole framework leading to failure of system B. The other would be that some part of the code works but the rest is not compatible with the system B.

3. Loss of Information

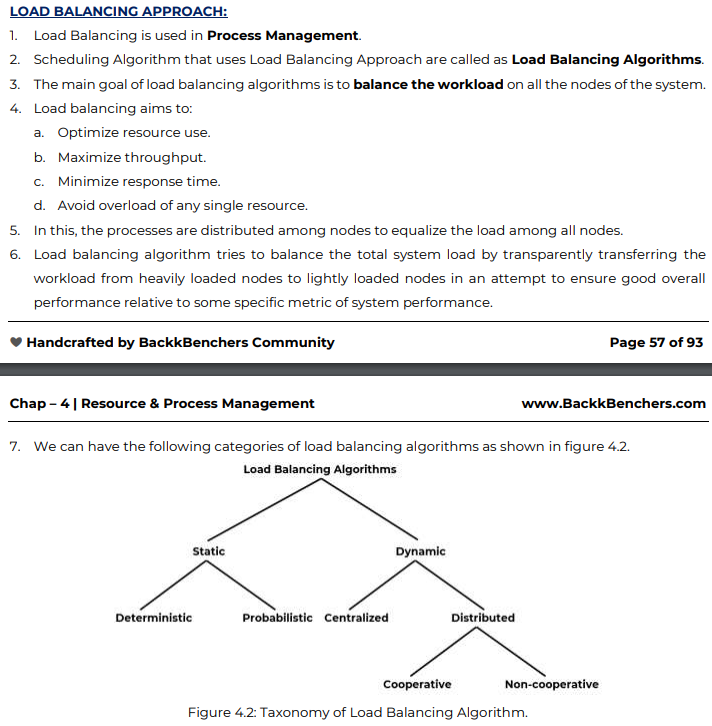
If during the migration of code from one system to another, the system B crashes due to some failure while handling other processes. It may happen that the migrated code will be lost. Hence, it is equally important to have the copy of the same data stored and to take steps in making sure that the transfer happens efficiently.

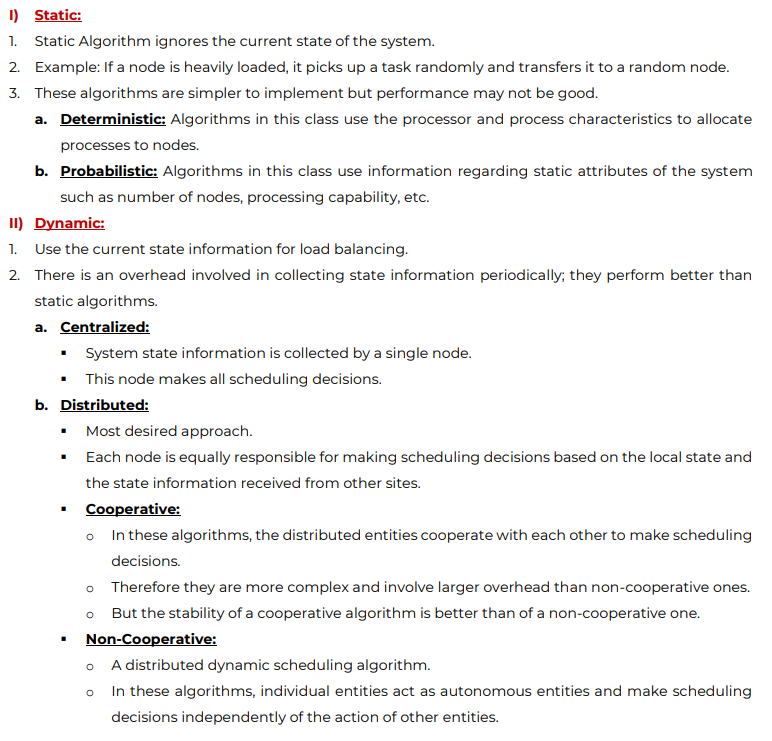
Adv :

* Increases Scalability
* Improves Performance
* Load Distribution
* Provides Flexibility

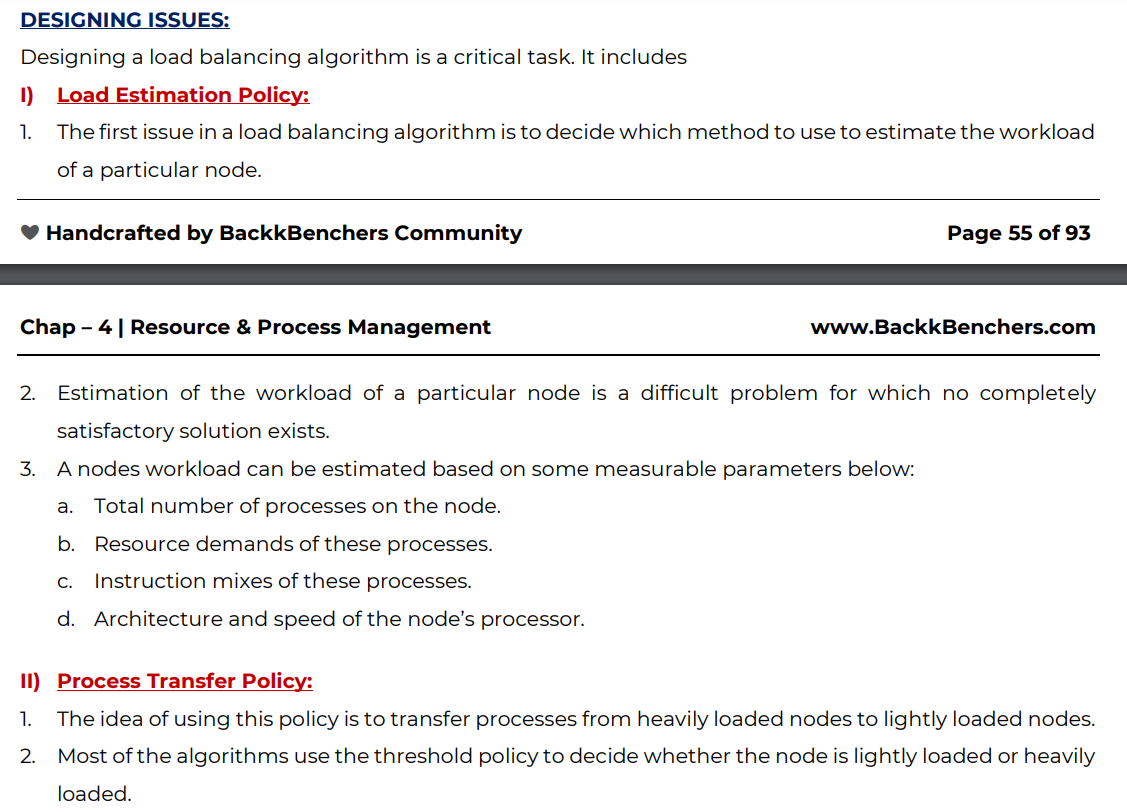
Load Balancing :

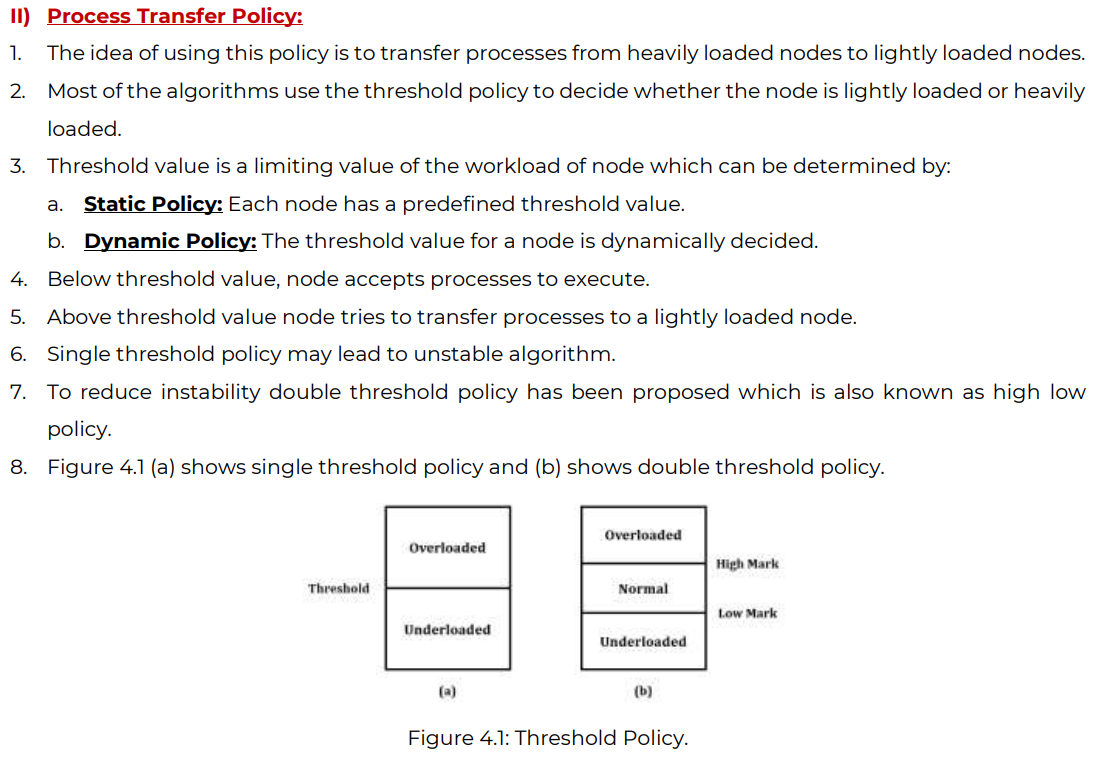
* What are the different types of load balancing algorithm/approach.

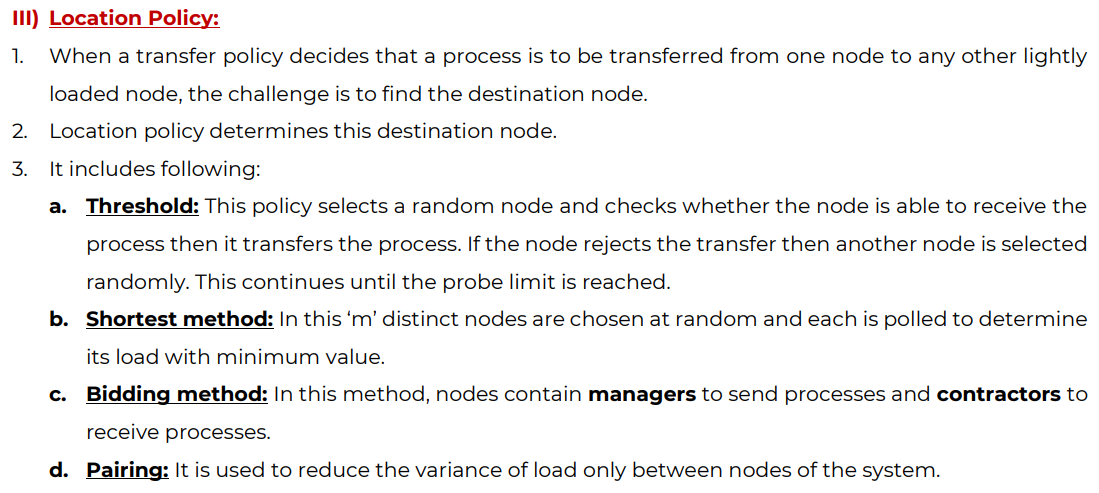


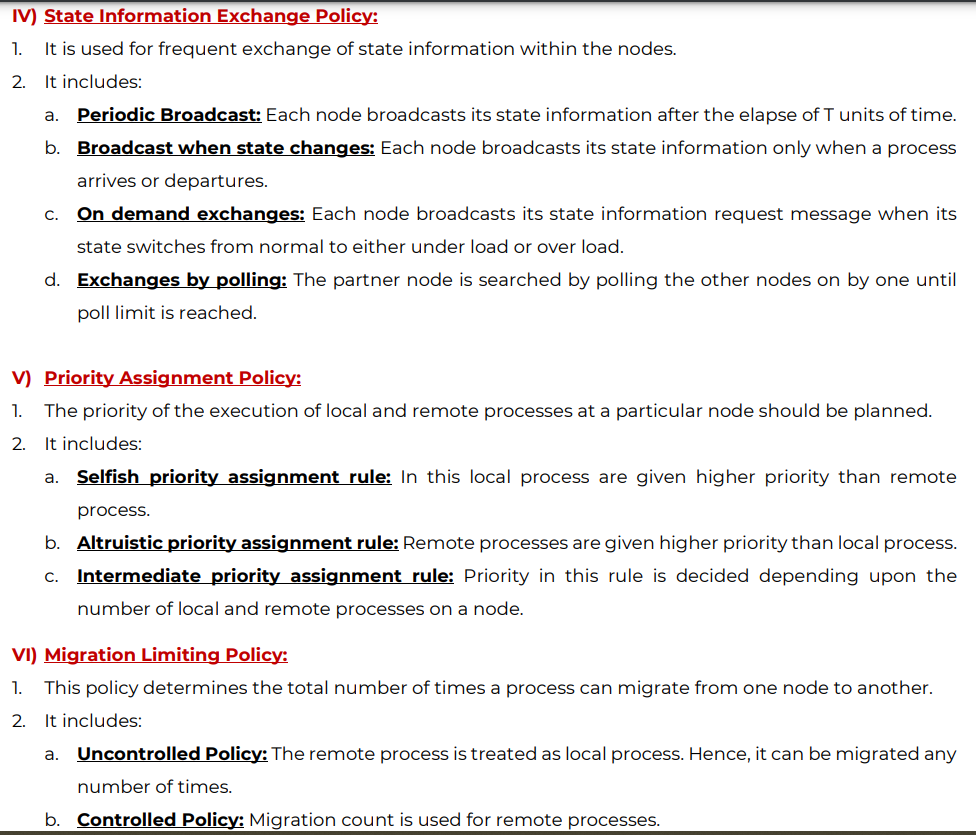


* Issues in LBA









Compare Load Sharing and Load Balancing strategies for scheduling processes in a distributed system

Compare and contrast Task Assignment, Load Balancing and Load Sharing approaches

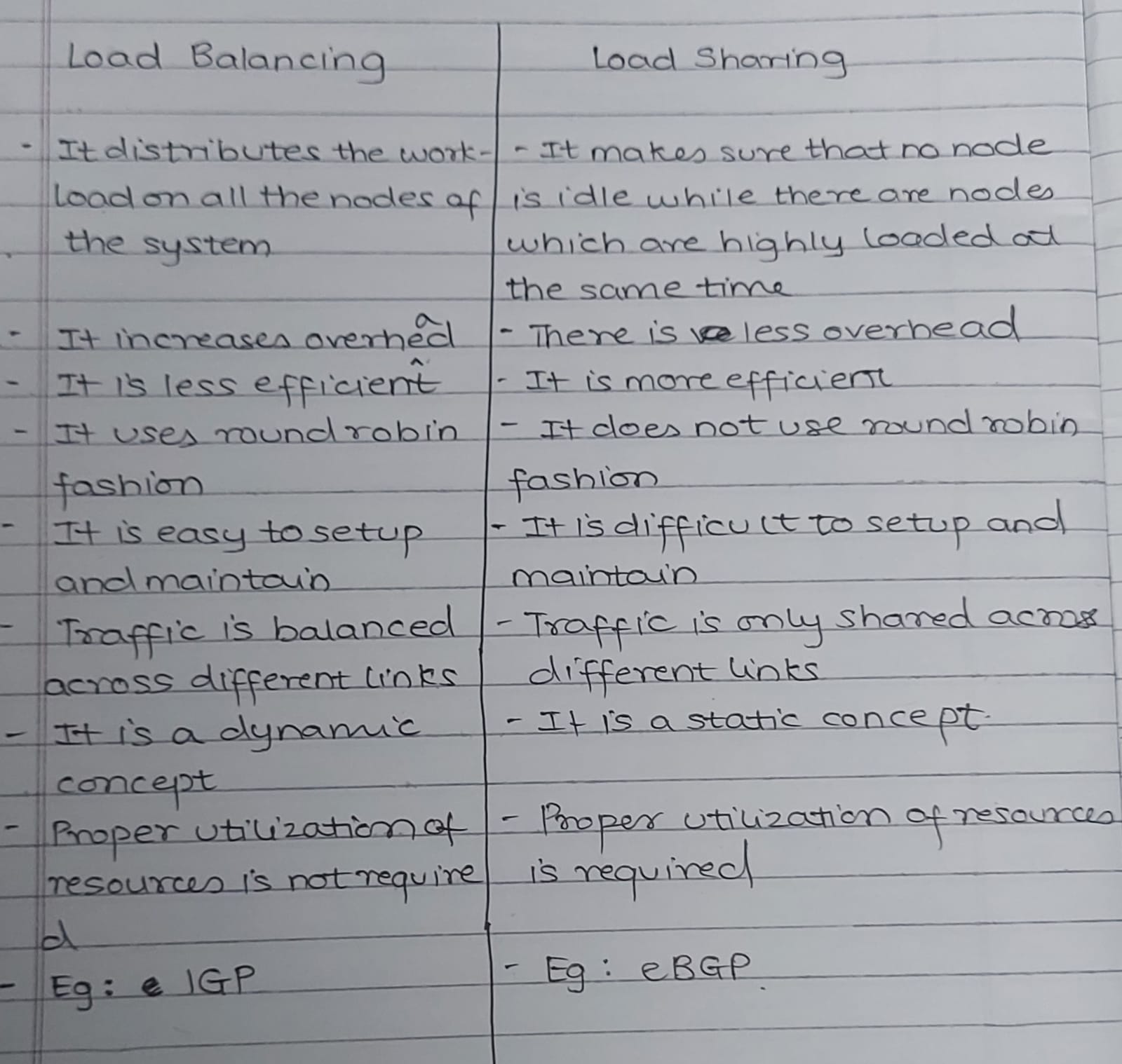
Compare Load Sharing to task assignment and load balancing strategies for scheduling processes in a distributed system.

Load Sharing:

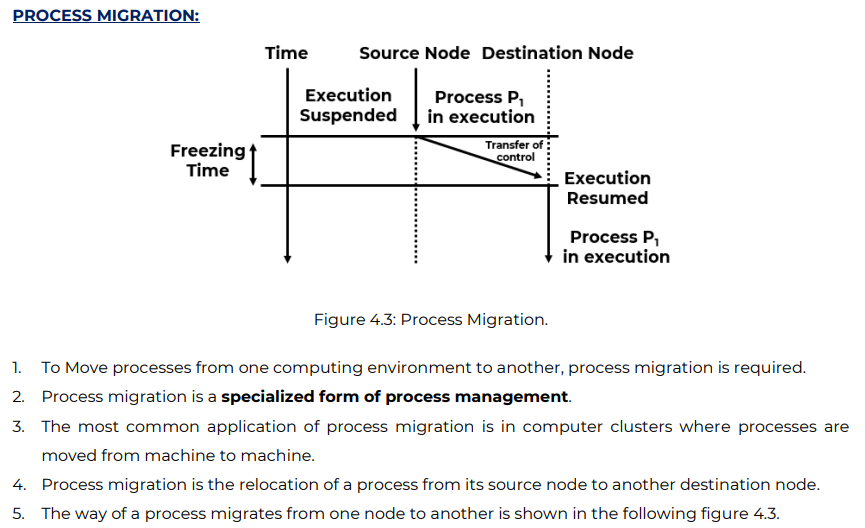
For proper utilization of the resources, it is not required that all the nodes should be balanced. It is necessary and sufficient to make sure that no nodes are idle while some nodes are heavily noded at the same time. This is called Load Sharing.

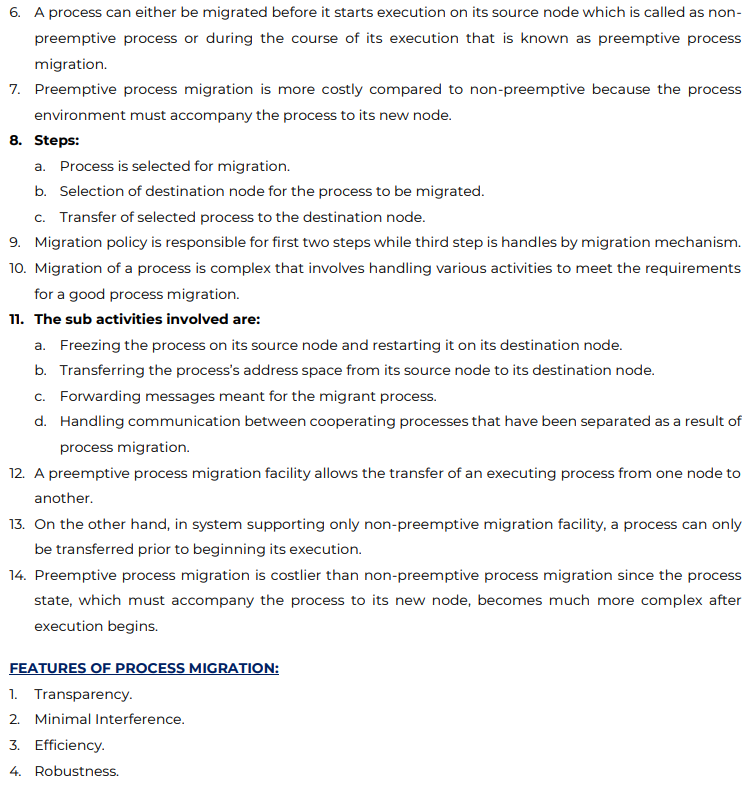
Task Assignment :

Each process is viewed as a collection of task. These tasks are then scheduled to suitable nodes. It is not a widely used approach because it requires characteristics of the processes before scheduling them. It also doesn’t take into consideration the current state of the system.



Discuss the need of process migration and the role of resource to process and process to resource binding in process migration.





Advantages of Process Migration :

1. Balancing the load

2. Reduce average response time

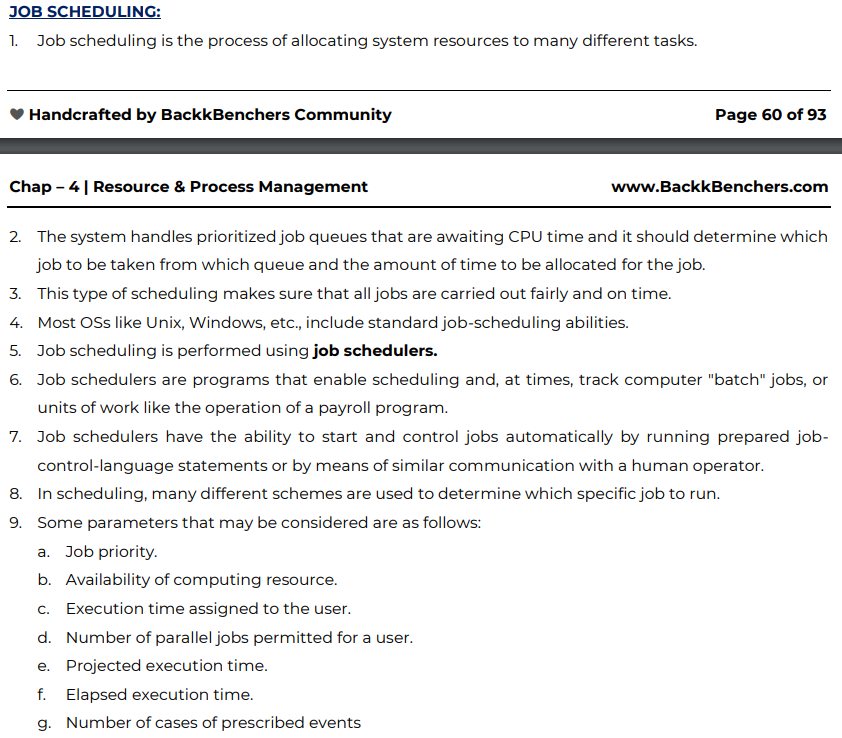
3. Speeds up individual jobs/tasks

4. Utilizing resources effectively

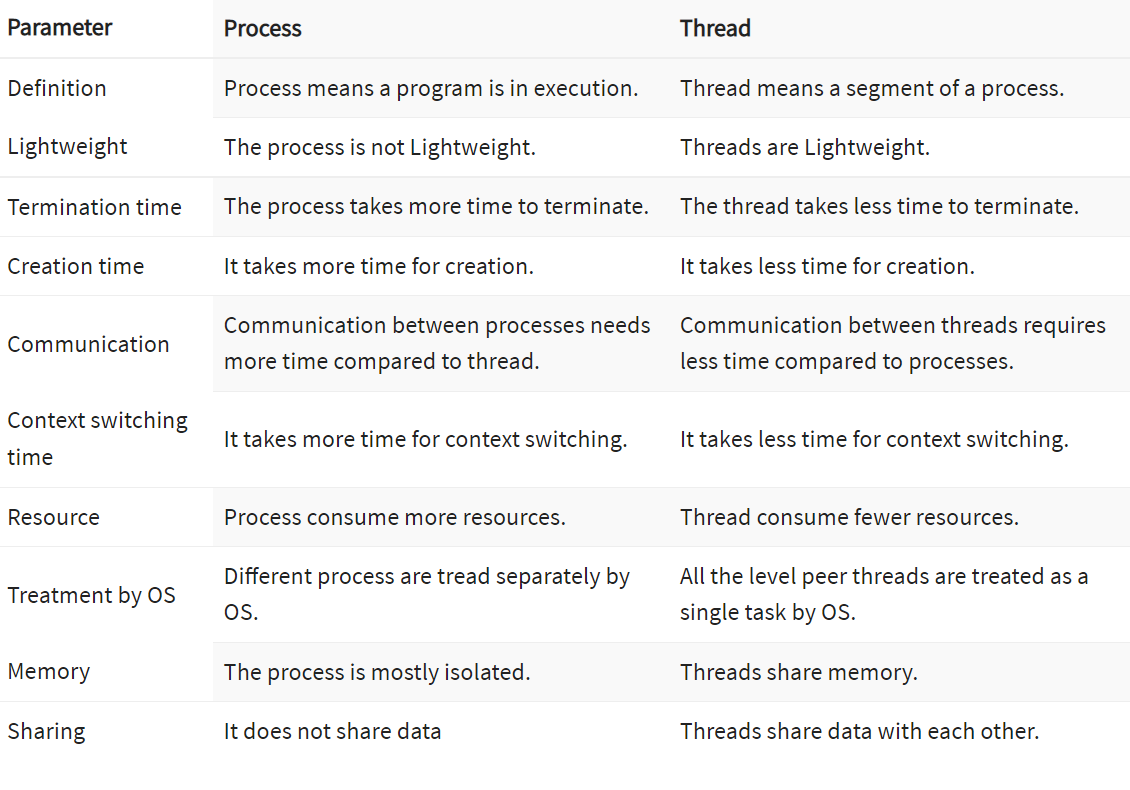
5. Reduce network traffic

Differentiate between job scheduling and load balancing.

Job Scheduling :



Difference between Process and Thread



What is Virtualization?

Virtualization is the creation of a virtual -- rather than actual -- version of something, such as an operating system (OS), a server, a storage device or network resources.

Virtualization uses software that simulates hardware functionality to create a virtual system. This practice allows IT organizations to operate multiple operating systems, more than one virtual system and various applications on a single server. The benefits of virtualization include greater efficiencies and economies of scale.

Types of Virtualization:

* Network
* Storage
* Server
* Data
* Desktop
* Application

Advantages:

* Lower Cost
* Easier Testing
* Improved Productivity

Benefits:

* Single Minded Servers
* Easy migration to the cloud

**Mod 5: Consistency, Replication and Fault Tolerance.**

1. Define Consistency Model. List its types

Consistency is the agreement between multiple nodes in a distributed system to achieve a certain value. It is to keep the replicas consistent. Informally, it means that when one copy of a file is updated, the copies of those files are also updated otherwise the replicas will not be same or not be consistent. Replication of the data improves reliability and performance. If one of the replicas of the file crashes, then the other replica is available for the job/task.

Types :

a. Strong Consistency Models

b. Weak Consistency Models

1. Client-Centric CMs

* Eventual Consistency
* Monotonic Reads
* Monotonic Writes
* Read your Writes
* Writes follow Reads

2. Data-Centric CMs

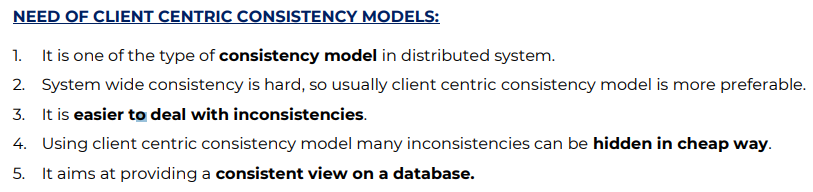
* Strict Consistency Model
* Sequential Consistency
* Linearizability
* Casual Consistency
* FIFO Consistency
* Weak Consistency
* Release Consistency
* Entry Consistency

Data Store : A distributed data store is a system that stores and processes data on multiple machines. As a developer, you can think of a distributed data store as how you store and retrieve application data, metrics, logs, etc

2. Explain Client Centric Models. Types. Provide suitable example application scenarios.

Clearly explain how monotonic read consistency model is different from read your write consistency model. Support your answer with suitable example application scenarios where each of them can be distinctly used.

Explain the need of client centric consistency models as compared to data centric consistency model.

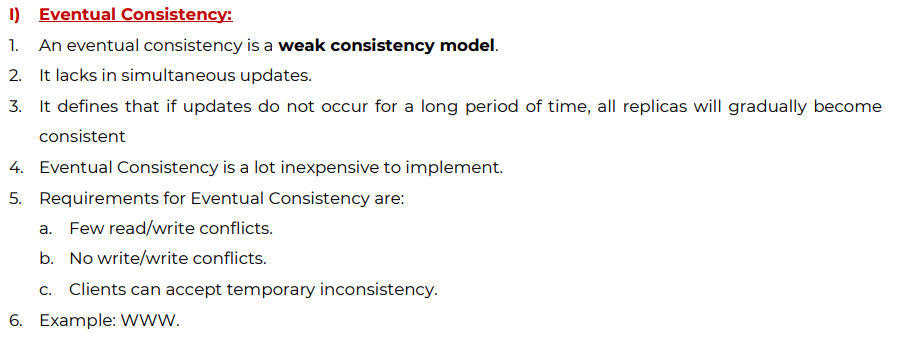


Client Centric Consistency Models:

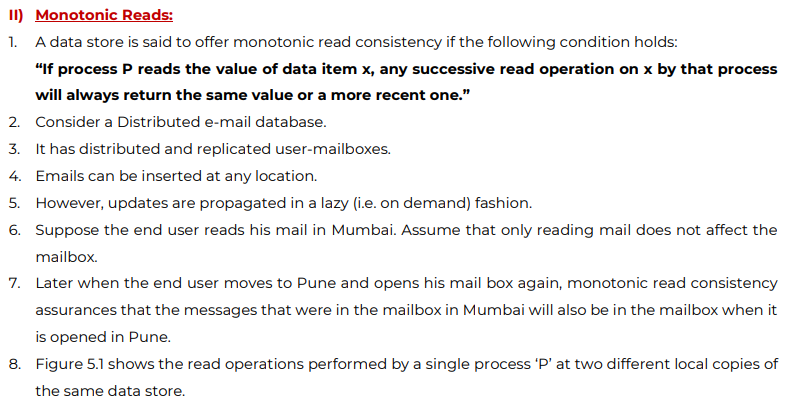
It concentrates on consistency from the perspective of a single client.

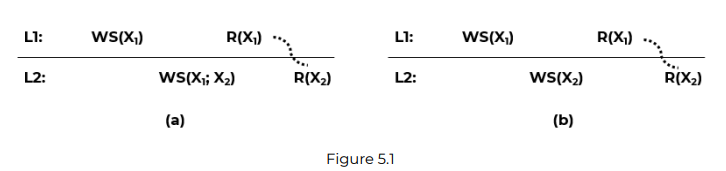
Client centric is an approach of doing business that focuses on the creating a positive experience for the customer. Client centric business ensures that the customer is at the center of the business philosophy, operations and ideas. Client centric business believes that their clients are the only reason that they exist and use every means at their disposal to keep the client happy and satisfied. It is also referred to as the customer centric.

* Eventual Consistency:



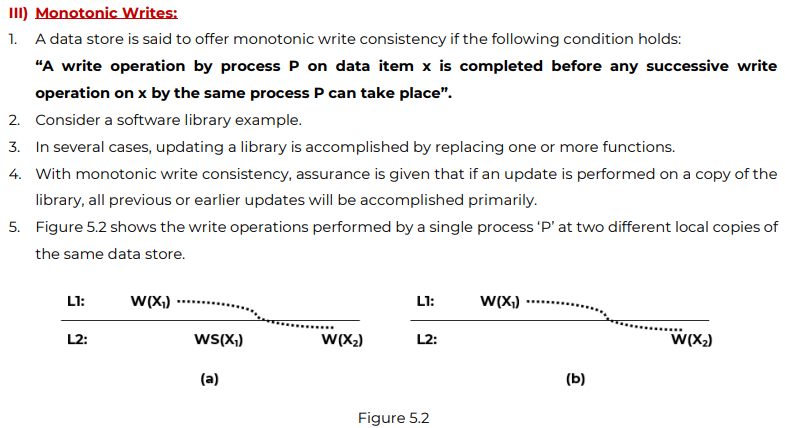
* Monotonic Reads:



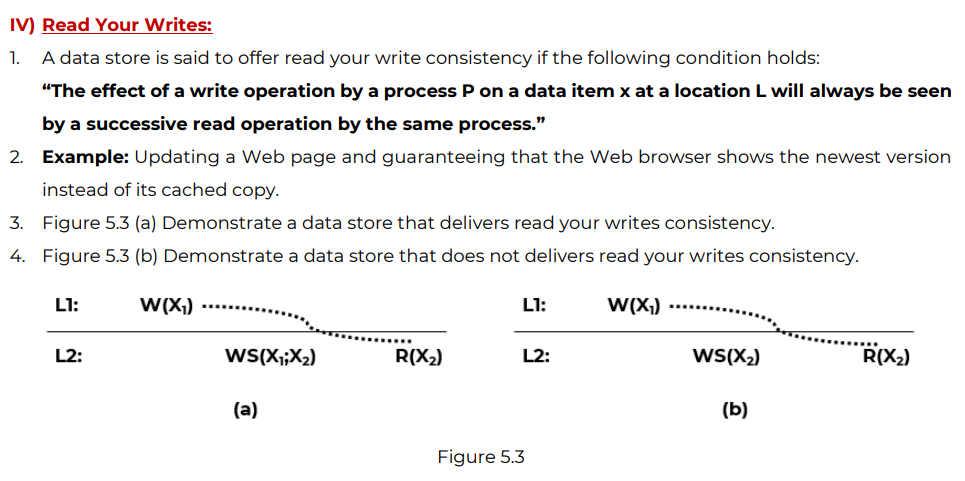


Example: Automatically reading your personal calendar updates from different servers.

* Monotonic Writes



* Read Your Writes

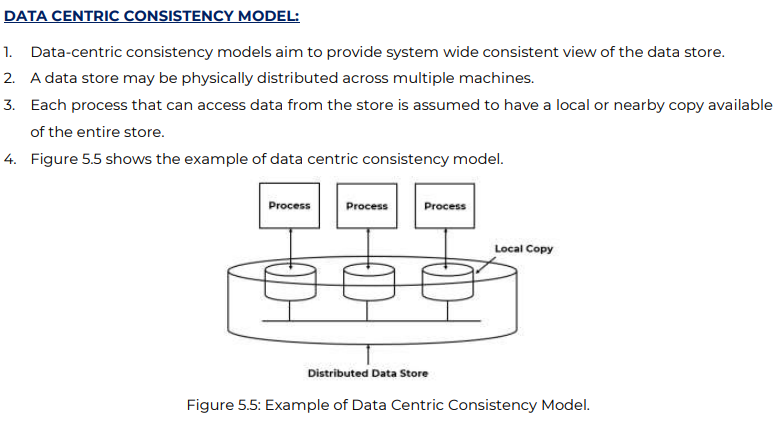


* Writes Follow Reads :

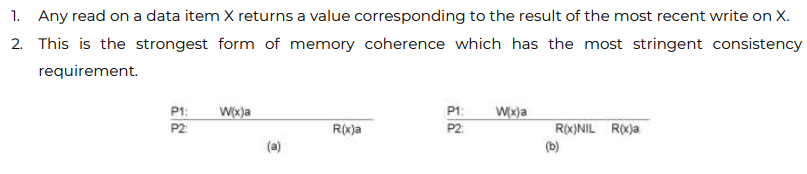
A write operation by a process P on a data item x following a previous read by the same process, is guaranteed to take place on the same or even a more recent valye of x, than the one having been read before.

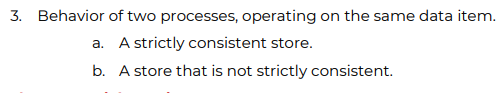
3. Data Centric Models. Types.

Data centric is used when there many clients.

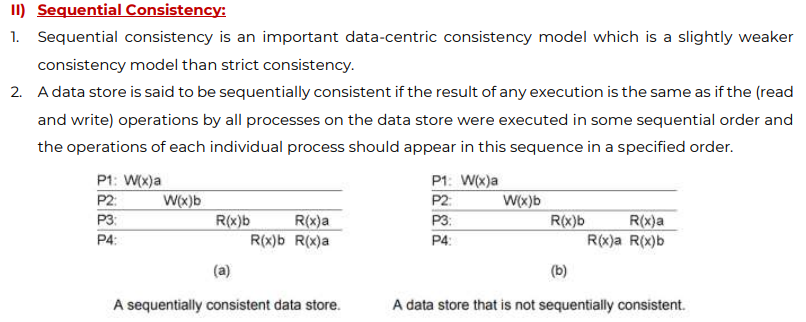


* Strict Consistency Model :

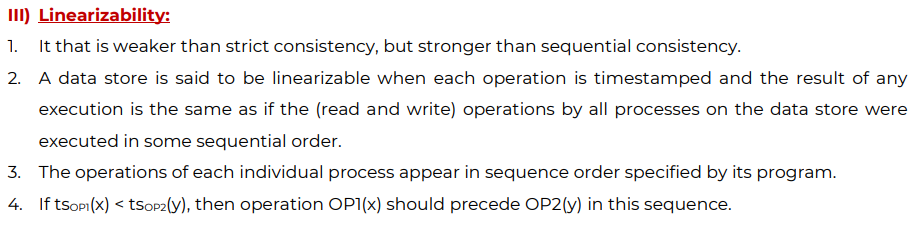




* Sequential Consistency:



* Linearizability :



* Casual Consistency

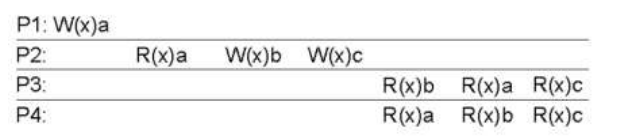
All processes see only those memory reference operations in the correct order that are potentially casually related.

Memory reference operations which are not related may be seen by different processes in different order.

* FIFO Consistency

It ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed like a single process in a pipeline.

This model is simple and easy to implement having good performance because processes are ready in the pipeline. Implementation is done by sequencing write operations performed at each node independently of the operations performed on other nodes.



* Weak Consistency

It enforces consistency on a group of memory reference operations rather than individual operations. A distributed shared memory system that supports the weak consistency model uses a special variable called a synchronization variable which is used to synchronize memory. When a process accesses a synchronization variable, the entire memory is synchronized by making visible the changes made to the memory to all other processes.

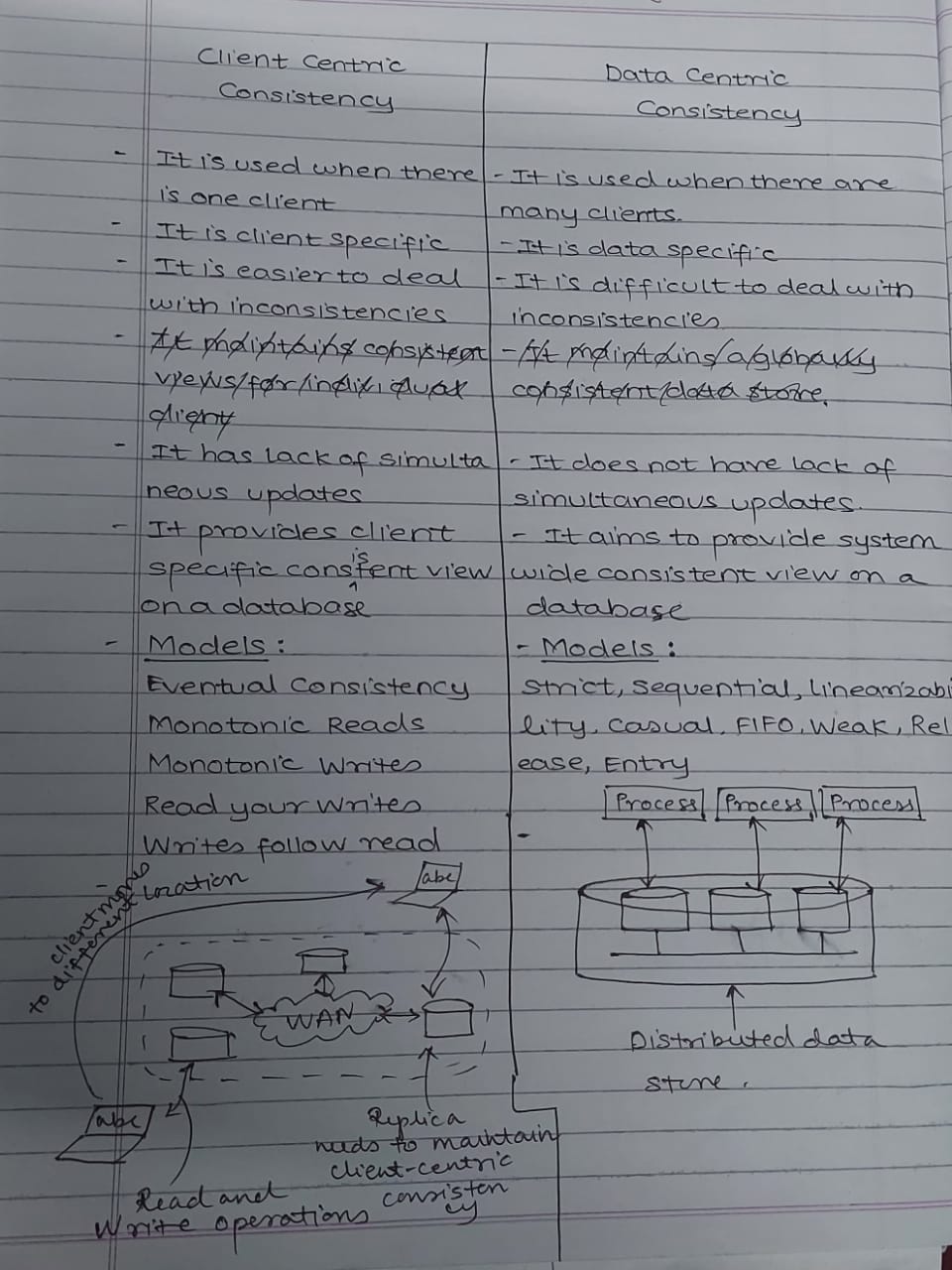
* Release Consistency

It tells whether a process is entering or exiting from a critical section so that the system performs either of the operations when a synchronization variable is accessed by a process. Two synchronization variables acquire and release are used instead of single synchronization variable. Acquire is used when process enters critical section and release is when it exits a critical section.

* Entry Consistency

Every shared data item is associated with a synchronization variable. In order to access consistent data, each synchronization variable must be explicitly acquired. Release consistency affects all shared data but entry consistency affects only those shared data associated with a synchronization variable.

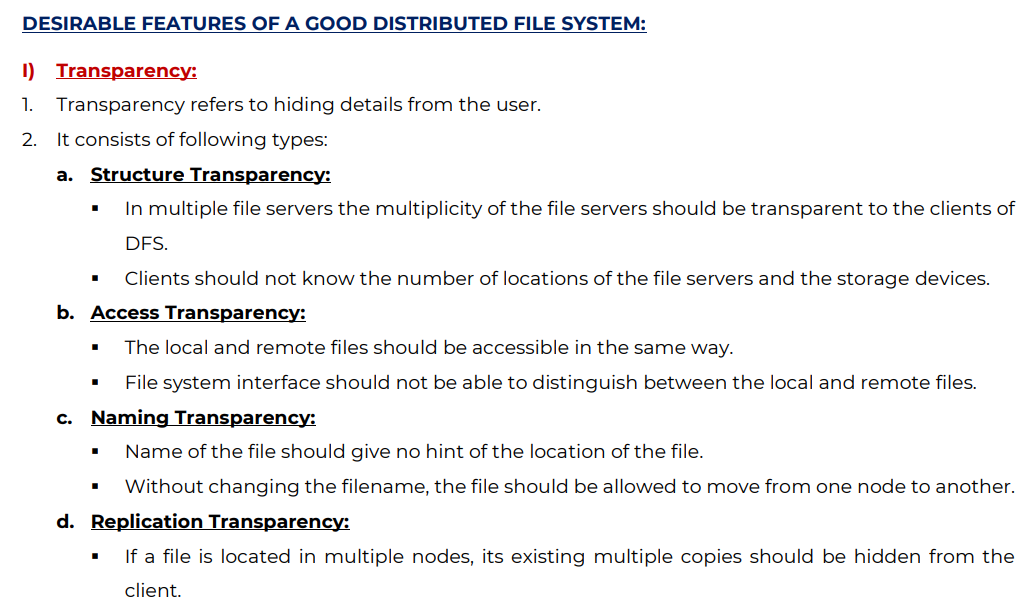
Difference between data centric and client centric consistency models.

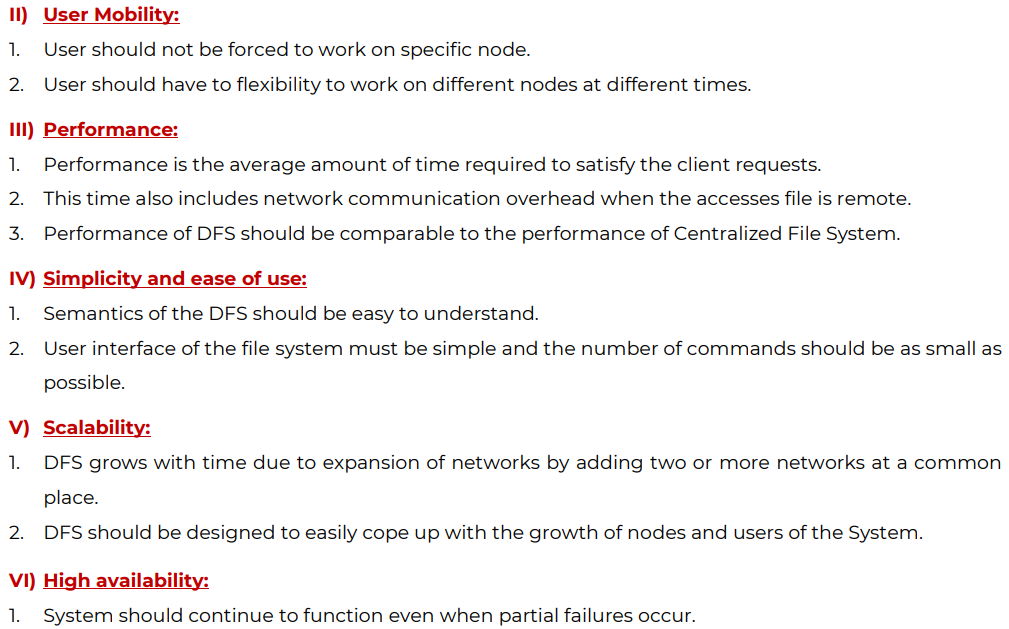


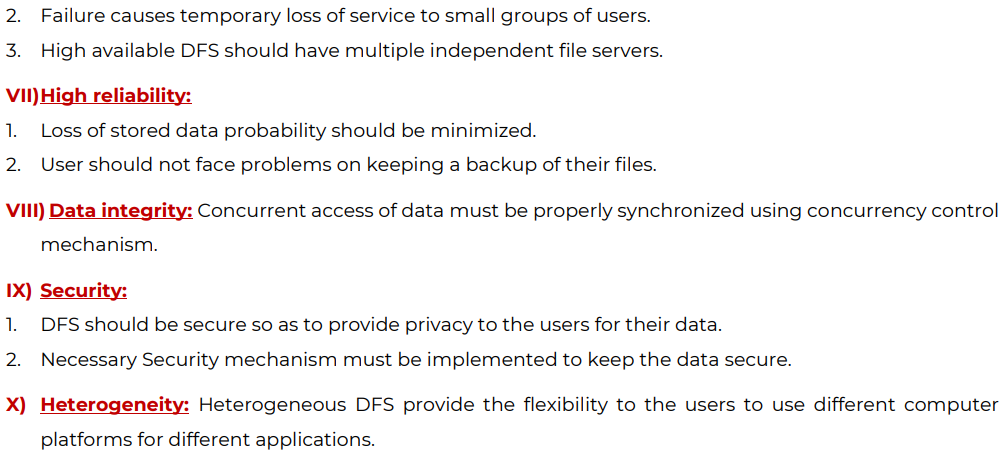
**Mod 6 : Distributed File Systems and Name Services**

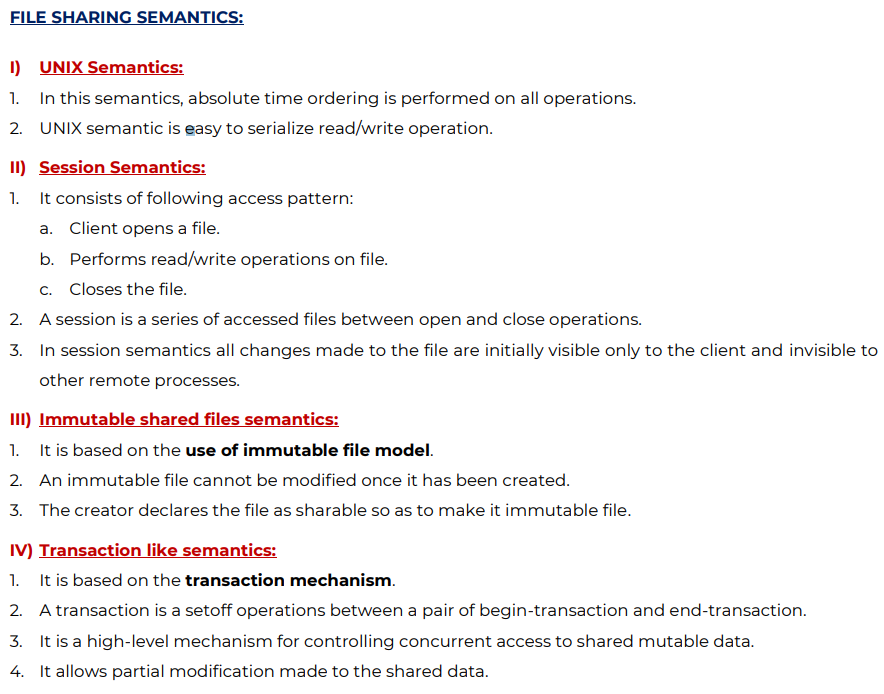
In computing, a distributed file system (DFS) or network file system is any file system that allows access to files from multiple hosts sharing via a computer network. This makes it possible for multiple users on multiple machines to share files and storage resources.

What are the good/desirable features of a distributed file system? Explain file sharing semantics of it.









What are the issues in designing a distributed system?

What are the common issues with which the designer of a heterogeneous distributed system must deal?

Designing issues of DS

· Heterogeneity

The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks. Internet consists of many different sorts of network their differences are masked by the fact that all of the computers attached to them use the Internet protocols to communicate with one another. For e.g., a computer attached to an Ethernet has an implementation of the Internet protocols over the Ethernet, whereas a computer on a different sort of network will need an implementation of the Internet protocols for that network.

Heterogeneity is applied to the network, computer hardware, operating system and implementation of different developers. A key component of the heterogeneous distributed system client-server environment is middleware. Middleware is a set of services that enables application and end-user to interacts with each other across a heterogeneous distributed system.

· Openness

The openness of a computer system is the characteristic that determines whether the system can be extended and reimplemented in various ways

The challenge to designers is to tackle the complexity of distributed systems consisting of many components engineered by different people

· Security

Many of the information resources that are made available and maintained in distributed systems have a high intrinsic value to their users. Their security is therefore of considerable importance.

Security of information system has three components Confidentially, integrity and availability. Encryption protects shared resources, keeps sensitive information secrets when transmitted.

· Synchronization/Concurrency

There is a possibility that several clients will attempt to access a shared resource at the same time. Multiple users make requests on the same resources, i.e. read, write, and update. Each resource must be safe in a concurrent environment. Any object that represents a shared resource a distributed system must ensure that it operates correctly in a concurrent environment.

· Failure Handling

Computer systems sometimes fail. When faults occur in hardware or software, programs may produce incorrect results or may stop before they have completed the intended computation. Failures in a distributed system are partial – that is, some components fail while others continue to function. Therefore, the handling of failures is particularly difficult.

· Scalability

Distributed systems operate effectively and efficiently at many different scales, ranging from a small intranet to the Internet. A system is described as scalable if it will remain effective when there is a significant increase in the number of resources and the number of users.

· Transparency

Transparency ensures that the distributes system should be perceived as a single entity by the users or the application programmers rather than the collection of autonomous systems, which is cooperating. The user should be unaware of where the services are located and the transferring from a local machine to a remote one should be transparent.

· Quality of Service

Once users are provided with the functionality that they require of a service, such as the file service in a distributed system, we can go on to ask about the quality of the service provided. The main nonfunctional properties of systems that affect the quality of the service experienced by clients and users are reliability, security and performance. Adaptability to meet changing system configurations and resource availability has been recognized as a further important aspect of service quality.

· Reliability

One of the original goals of building distributed systems was to make them more reliable than single-processor systems. The idea is that if a machine goes down, some other machine takes over the job. A highly reliable system must be highly available, but that is not enough. Data entrusted to the system must not be lost or garbled in any way, and if files are stored redundantly on multiple servers, all the copies must be kept consistent. In general, the more copies that are kept, the better the availability, but the greater the chance that they will be inconsistent, especially if updates are frequent.

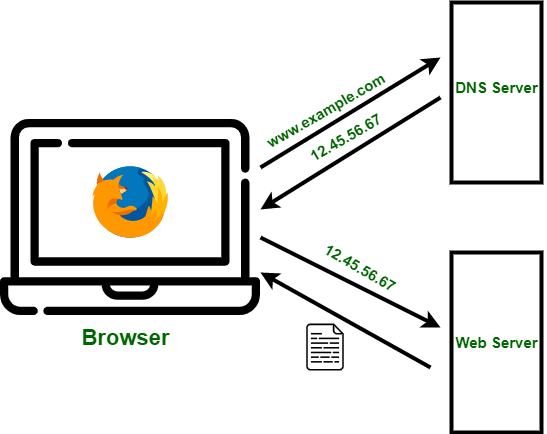
· Performance

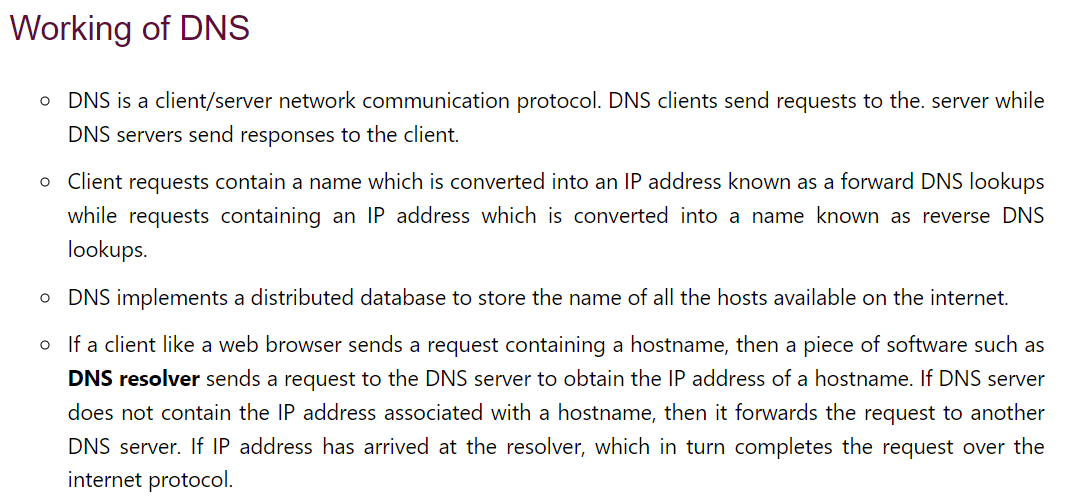
The performance of the distributed system should be same as that of centralized system. It is measured as the average amount of time needed to satisfy the client requests.

Explain Domain Name System.

The Domain Name System (DNS) is the Internet's system for mapping alphabetic names to numeric Internet Protocol (IP) addresses like a phone book maps a person's name to a phone number. For example, when a Web address (URL) is typed into a browser, a DNS query is made to learn an IP address of a Web server associated with that name.

Using the www.example.com URL, example.com is the domain name, and www is the hostname. DNS resolution maps www.example.com into an IP address (such as 192.0.2.1). When a user needs to load a webpage, a conversion must occur between what a user types into their web browser (www.example.com) into an IP address required to locate the www.example.com site.





Generic Domain :

* com : commercial organizations
* Edu : educational institutions and universities
* gov : governmental agencies
* mil : military organizations
* int : international organizations

Country Domain :

* us : united states
* fr : France
* in : India

Short Note: Andrew File System (AFS)

AFS is a distributed file system.  It uses the client/server model, where all the files are stored on file server machines.  Files are transferred to client machines as necessary and cached on local disk.  The server part of AFS is called the AFS File Server, and the client part of AFS is called the AFS Cache Manager.  AFS provides Access Control Lists (ACLs) which provide for more control and flexibility than standard Linux file permissions.

AFS provides transparent access to local and remote files by using a consistent name space. All files in AFS are found under the Linux directory /AFS.

Operations:

* l (lookup) – list files in the directory (but subdirectories are controlled by their own ACLs)
* i (insert) – create files in the directory
* d (delete) – delete files and subdirectories from the directory
* a (administer) – change the ACL for the directory
* r (read) – read files in the directory
* w (write) – modify files in the directory and modify their UNIX mode bits with the chmod command
* k (lock) – run programs that need to “flock” files in the directory

Components of AFS:

AFS is implemented as two software components which exists in a UNIX environment as processes called Vice and Venus:

* Vice: It is a server-side process that provides shared file services to each client.
* Venus: It is a client-side cache manager which acts as an interface between the application program and the Vice.

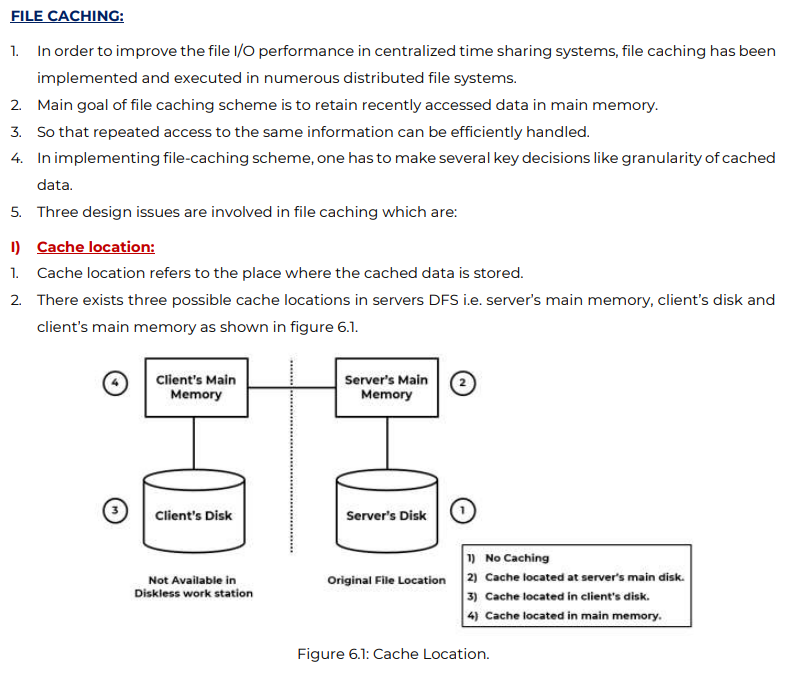
Advantages of AFS :

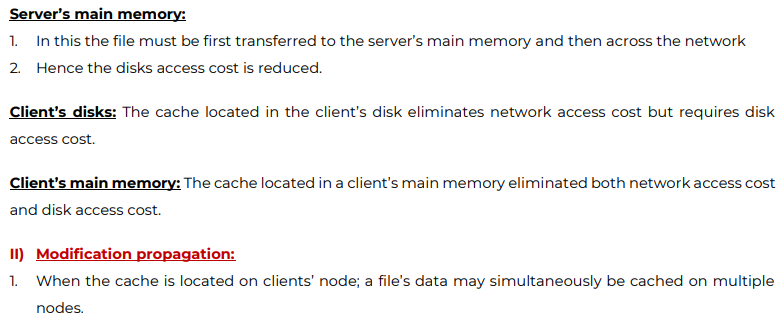
* It sets up a lot of storage space for caching
* The files that aren’t updated very often last a long time

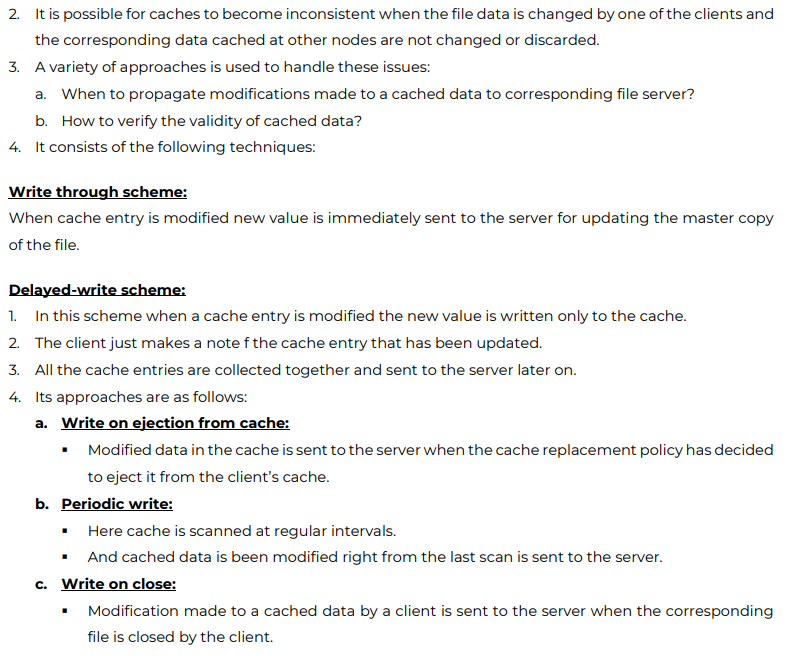
Limitations of AFS :

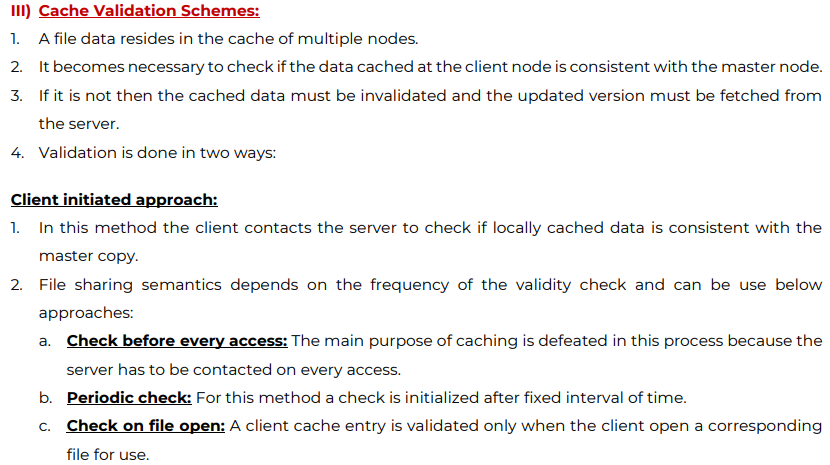
* It fails to meet the services when any components in the network fail.
* Scalability problem.

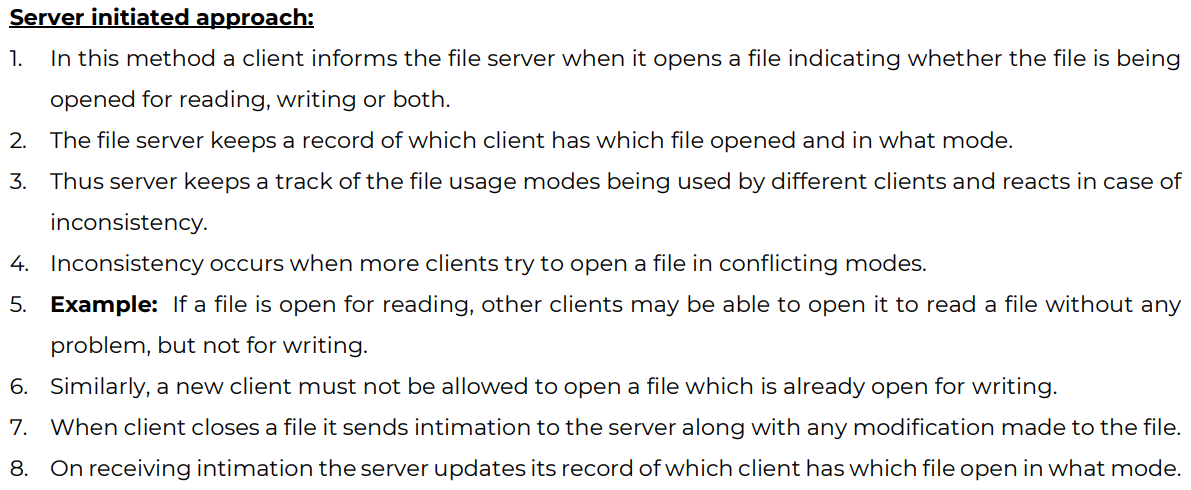
**Discuss File Caching for Distributed Algorithm/Schemes.**



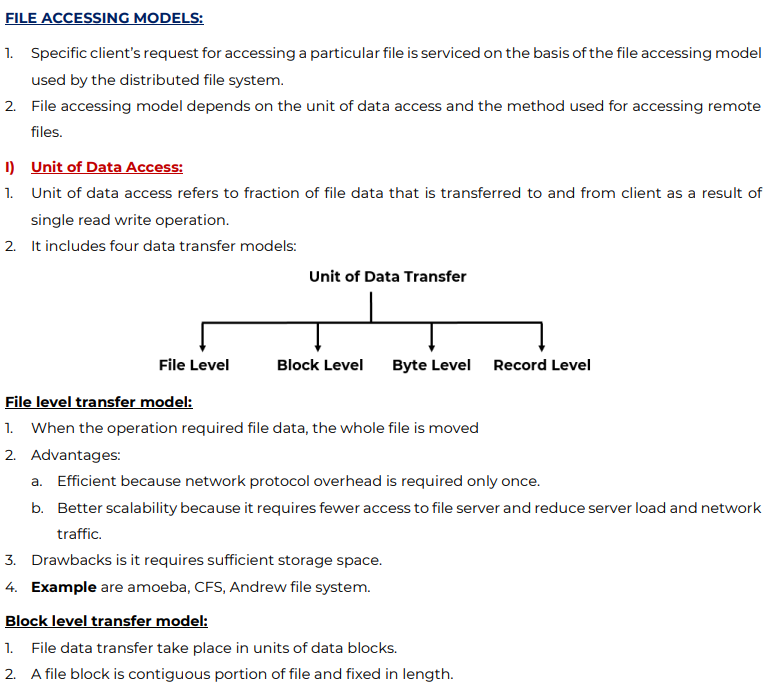


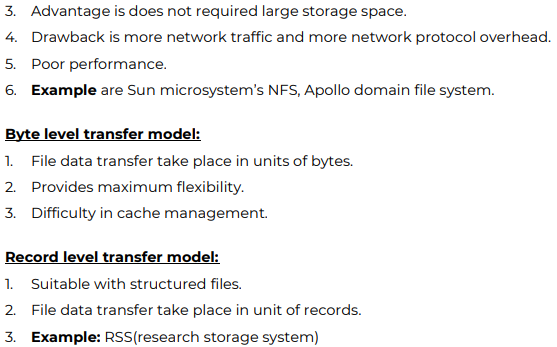


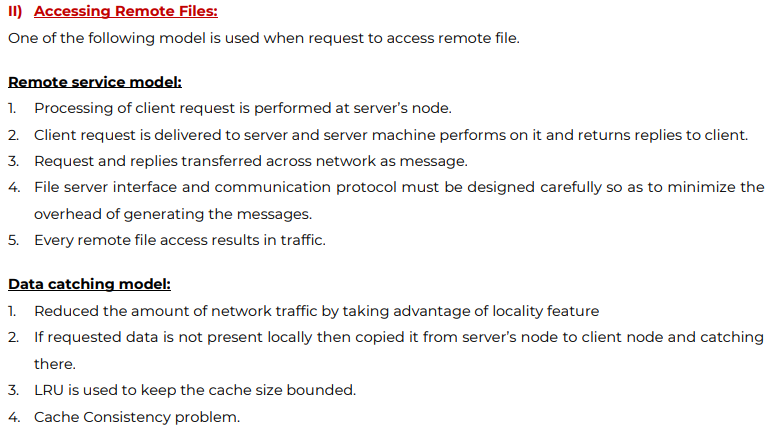




Explain File Accessing Methods







Explain Name Resolution.

